VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

2D katapultovací hra v prostředí Internet

2D Catapulting Game over Internet

2015 Lukáš Vnenk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Vnenk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: 2D katapultovací hra v prostředí Internet
2D Catapulting Game over Internet

Zásady pro vypracování:

Cílem práce je vytvořit hru v prostředí Internetu, která umožní připojit hráče prostřednictvím klientské aplikace na platformě JavaSE a na platformě Android. Hlavním motivem hry je katapultování postavičky hráče, která se snaží doletět co nejdále. Postavička v průběhu letu bude konfrontována s různými objekty, které můžou ovlivnit její rychlost a směr (případně další vlastnosti). Hra bude obsahovat i speciální objekty, které umožní hrát mini-hry.

Hra bude umožňovat:
1. Zobrazování hry ve 2D pohledu.
2. Hrát vložené mini-hry.
3. Souboj více hráčů - kdo dále doletí.
4. Připojení z platformy JavaSE a Android.
5. Program umožní připojení do současně vyvíjeného portálu her v jazyce Java.

Práce bude obsahovat:
1. Implementaci výše popsané hry.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2015 .....

Rád bych na tomto místě poděkoval panu Ing. Davidu Ježkovi a všem dalším, kteří mi s prací pomohli, za jejich vstřícnost, připomínky a cené rady.

Abstrakt

Cílem práce je vytvořit 2D hru s možností hraní přes internet, ve které je hlavním cílem co nejvyšší dosažená vzdálenost katapultem vymrštěné postavičky. Ta se v průběhu hry střetává s různými překážkami, které ovlivňují její let. Součástí hry jsou také minihry, jejichž úspěšné plnění pomáhá hráči získat rychlost a dostat se tak dále. Veškeré herní animace jsou vytvořeny v aplikaci Spine. Jedná se o animaci pomocí koster a klíčových snímků. Hra je implementována ve frameworku LibGDX, který je zaměřen na vývoj v jazyce Java. V první části práce řeším implementaci hry a technologie použité při jejím vývoji, ve druhé programátorskou dokumentaci s využitím diagramů jazyka UML a ve třetí uživatelskou dokumentaci.

Klíčová slova: Android, Java, LibGDX, 2D Katapultovací hra

Abstract

Goal of this bachelor is to develop 2D game with the possibility of playing over Internet, in which the main goal is to reach the biggest possible distance with catapulted character. During the gameplay, the character is confronted with different obstacles, which affects his flight. Part of game are also minigames. Successful fulfillment of minigames helps player to gain more speed and get further. All of the animations are done using Spine application, which works with skeletons and keyframes. The game is implemented in LibGDX framework for Java. In the first part I deal with implementation and technologies used to develop the game, in second part there is programmer documentation with use of UML diagrams and third part is user documentation.

Keywords: Android, Java, LibGDX, 2D Catapulting game

Seznam použitých zkratek a symbolů

GUI	–	Graphical User Interface
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
FPS	–	Frames Per Second

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Hra, kterou jsem nazval **Bobo**, je originální akční hra, mířená především pro mobilní zařízení. Je zobrazována z bočního 2D pohledu a napsána v jazyce **Java** s použitím rámce **libGDX** a technologie skeletonových animací.

Cílem hry je osvobodit hlavního hrdinu ze spárů nepřátelského kmene odpálením z katapultu. Hrdina letí vzduchem, odráží se od země, nepřátel apod. Hráč nad ním má kontrolu díky vzdušným úderům, které vždy hrdinu pošlou vysokou rychlostí střemhlav k zemi, kde odrovná nejbližší nepřátele, odrazí se a získá rychlost. Zpočátku však katapult nemá dostatečnou sílu na to, aby dostal hrdinu přes všechny nepřátele až k poslední bráně vedoucí na svobodu. Proto je nutné pravidelně nejen katapult vylepšovat.

Druhů vylepšení je spousta a jsou velmi důležité. Každé z nich má navíc 7 úrovní a ovlivňuje průběh hry úplně jinak. Jako příklad uvádím jedno z vylepšení zvané Pružnost, díky kterému se s každou úrovní tohoto vylepšení stává hrdina pružnější a více se odráží od země, takže se dostane v každém dalším kole dále.

Hra umožňuje také souboj více hráčů přes internet. Hráči mají předem stanovený obnos herní měny, za který si před odpálením nakoupí libovolné vylepšení podle vlastní strategie. Poté se utkají v tom, kdo se dostane dále.

Hru doprovází množství miniher, jejichž plněním, získává hrdina na rychlosti podle toho, jak je hráč v dané minihře úspěšný. Minihry je možné spustit přímo ve hře po odpálení katapultem a nalezení speciálních nepřátel, kteří minihry představují.

V první kapitole s názvem **Programátorská dokumentace** se věnuji popisu technologií, které jsem použil při tvorbě hry, ukázkám implementace zajímavých metod a částí kódů s jejich popisem, použitým návrhovým vzorům a také diagramům v jazyce UML.

V druhé kapitole s názvem **Uživatelská dokumentace** směřuji přímo k hráči popis celého grafického uživatelského prostředí, popis veškerých pravidel, hlavních herních principů, ovládní hry apod.

2 Programátorská dokumentace

V této části bakalářské práce se věnuji postupům, řešením a technologiím, které jsem využil při vývoji hry.

2.1 Použité technologie

2.1.1 libGDX

LibGDX[1] je multiplatformní rámec určený předně pro vývoj her v jazyce Java. V současné době podporuje vývoj pro Android, iOS, Windows, Linux, Mac OSX a HTML5.

Rámec má otevřenou (open source) licenci a pracuje na něm pravidelně v současné době přibližně 300 lidí. Vývoj jde stále kupředu a on se tak neustále rozšiřuje a zlepšuje. Drobné opravy a vylepšení jsou téměř na denním pořádku.

Díky libGDX stačí napsat kód aplikace pouze jednou a on už se postará o její export na více platforem, aniž by bylo nutné kód měnit, což je obrovská výhoda. Vývoj aplikace probíhá většinou v prostředí stolního počítače například v programu Eclipse, který v sobě obsahuje vše potřebné pro práci v Javě, a také se do něj dají přidat moduly pro debugování na mobilních zařízeních v reálném čase. To všechno podstatně zrychluje vývoj.

LibGDX je velmi komplexní rámec a kromě pokročilých zobrazovacích metod také dovoluje zasahovat do systému souborů, vstupních i výstupních zařízení, zvukových zařízení apod. Obsahuje také množství API, které pomáhají se základními prvky tvorby hry, jako je například zobrazování grafiky (obrázků nebo sekvencí obrázků), vykreslování textu, tvorbu rozvržení a vzhledu uživatelského rozhraní, přehrávání zvukových efektů nebo hudebního doprovodu, matematické i geometrické výpočty, práce se soubory JSON, XML a tak dále.

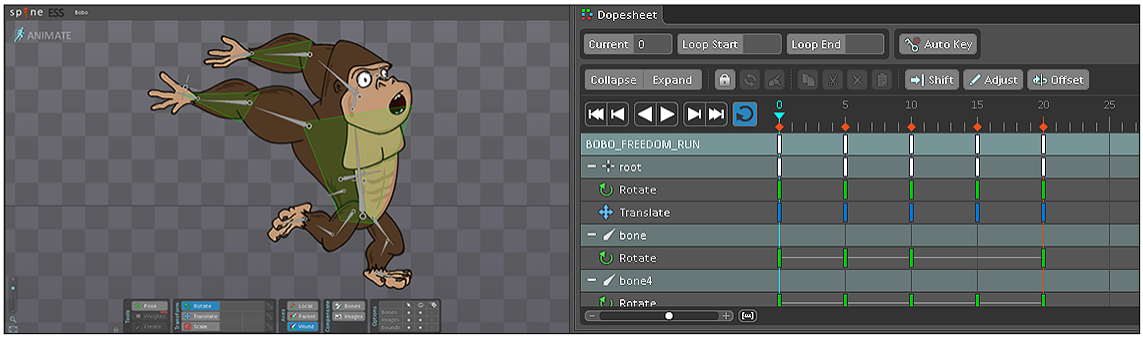
LibGDX také zná slabiny každé z podporovaných platforem, a tak je obchází svým vlastním způsobem. Veškerá tato funkcionalita je zabudována a skryta v jádru libGDX, a tak se o ně vývojář nemusí zajímat.

V dnešní době existuje také několik multiplatformních enginů pro vývoj her, libGDX se však snaží zůstat pouze rámcem, díky čemuž si může vývojář sám vybrat styl, kterým bude hru vyvíjet.

Pro tvorbu grafického prostředí je zde připraven systém Scene2D, který jej umožňuje tvořit pomocí logické tabulky a nastavení jejich jednotlivých buňek, což přináší mnoho výhod, co se týče ukotvení, zarovnání apod. (viz sekce Tvorba grafického rozhraní 2.5)

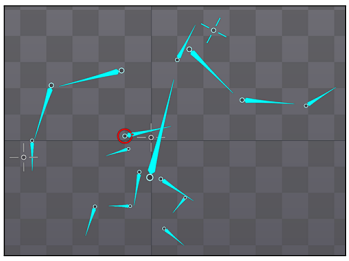
2.1.2 Spine

Spine[2] je software určený ke tvorbě 2D animací zejména do her. Má opravdu nádherné a intuitivní prostředí, díky čemuž je práce s ním rychlá a dokonce se ani nejeví jako práce, ale spíše jako zábava. Animace využívají systém kostí a klíčových snímků. Jsou interpolované, což zajišťuje jejich naprostou plynulost, při dosažení rozumných FPS (snímků za sekundu) v aplikaci, do které jsou vloženy.



Obrázek 1: Program Spine

Nejprve je potřeba nadefinovat kostru objektu, který budu chtít animovat. Kostra má vždy základní kost tzv. **root**, od které se odvíjí veškeré další kosti. Každá kost musí mít kost nadřazenou, které když změním vlastnosti, ovlivním tím i všechny kosti podřazené. Tím se zajistí konzistence celé kostry. Na každou kost hotové kostry poté můžu přidat **slot**. Slot můžu přirovnat ke složce, do které nahraju množství obrázků s tím, že je vždy aktivní a na dané kosti proto viditelný jen jeden. Slot kopíruje veškeré transformace kosti, na které je vytvořen, a tak při pohybu kostí hýbu i s připojeným obrázkem.



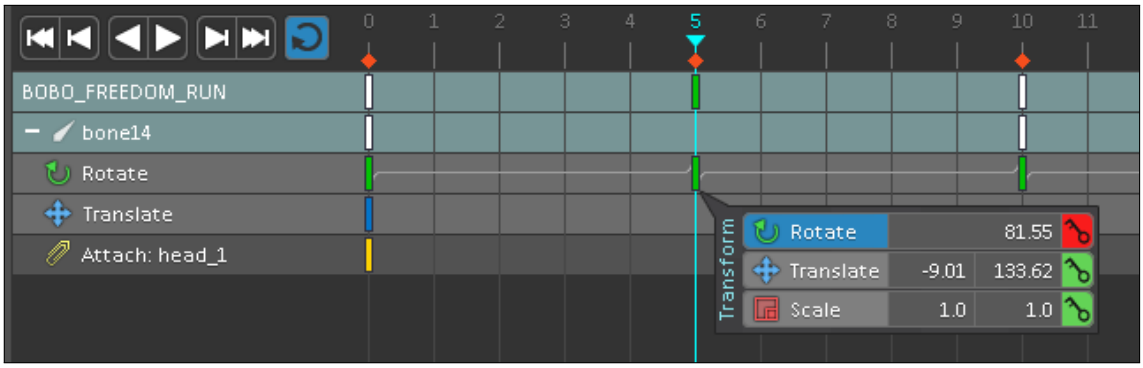
Obrázek 2: Kostra

Obrázek 3: Kostra 2

Po vyhotovení kostry a připojení obrázků, mám hned možnost vidět, jak bude výsledná postavička (nebo jakýkoliv jiný předmět animace) vypadat. V této fázi je vhodné napozicovat obrázky vzhledem ke kostem do finální podoby, abych se mohl rovnou přesunout k samotnému animování. K úpravě vlastností kostí a pozicování obrázků se ale mohu kdykoliv vrátit.

Nyní může započít proces animování postavičky. Spine obsahuje speciální animační mód, po jehož aktivaci se uživatelské prostředí pozmění pro tyto účely. Zobrazí se velký panel pro práci s klíčovými snímky, přehrávání animace a její nastavení. Součástí tohoto panelu je také záložka s prostředím pro editaci interpolace vybrané animace.

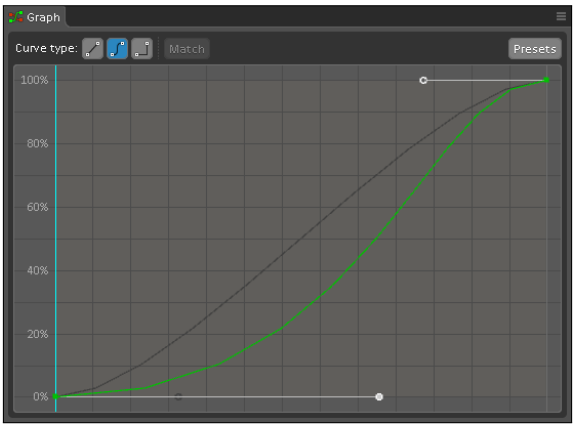
Tvorba animací v programu Spine funguje tak, že se nejprve provede změna pózy postavičky nebo některé z jejich částí. Poté se výsledné rozložení kostí zapíše jako transformace do klíčového snímku. Do klíčových snímků lze uložit změnu pozice, rotace a velikosti jednotlivých kostí.



Obrázek 4: Klíčové snímky

Vytvořené klíčové snímky se dají velice snadno editovat. Stačí na některý z nich kliknout, přenastavit transformace kostí podle potřeby a snímek znova uložit. Snímku bude i po změně transformací zachována případná interpolace vzhledem k okolním klíčovým snímkům. Lze je označovat obdélníkovým výběrem, libovolně kopírovat i vkládat se všemi jejich vlastnostmi a samozřejmě také mazat.

Může být použita **interpolace**, která určuje průběh změny transformací mezi klíčovými snímky. Bez interpolace animace skáče z jednoho klíčového snímku na další a výsledek tak není plynulý. Se zapnutím interpolace mezi klíčovými snímky se odemykají další možnosti nastavení. Hlavní je typ interpolace - **lineární** a **Bezierova**. Lineární interpolace zajišťuje průběh animace konstantní rychlostí. Bezierova interpolace lze nastavit křivkou, která umožňuje změnu rychlosti v průběhu animace.

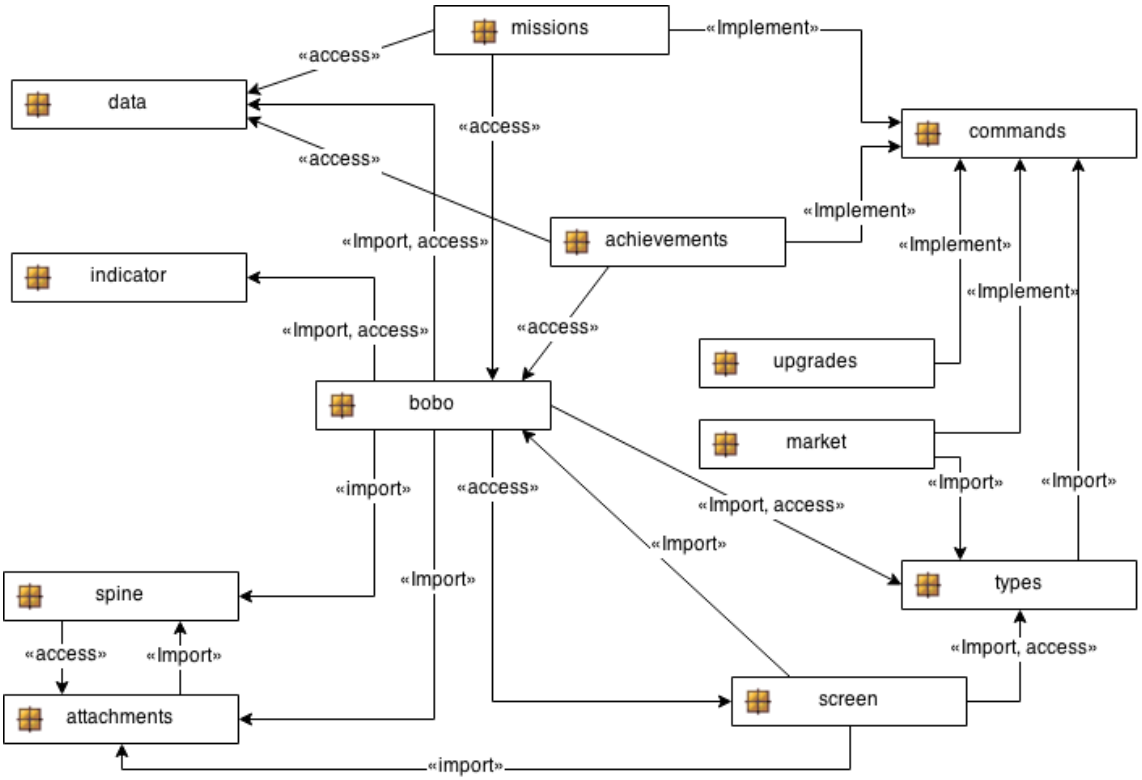


Obrázek 5: Graf interpolace - Bezier

Takto vytvořené skeletony i s animacemi jsou připraveny k exportování, pro které má Spine velké množství nastavení. Předně je možné nastavit, zda má být soubor s animacemi binární, nebo JSON (jenž jsem použil při vývoji). Dá se nastavit také kvalita výstupní textury, její formát a mnoho dalších věcí. (více v sekci 2.11)

2.2 Struktura projektu

Z důvodu velkého množství tříd a rozdělení projektu do několika menších částí, zde pro lepší orientaci uvádím diagram balíků (package diagram).



Obrázek 6: Struktura projektu - diagram balíků

Projekt je vytvořen v programu Eclipse s přídavnými zásuvnými moduly pro podporu Android a skládá se z více podprojektů. Jeden podprojekt pro každou platformu, na které je hra spustitelná.

Hlavní projekt s názvem **MadBobo** obsahuje veškeré zdrojové kódy k vytvoření grafického uživatelského rozhraní, funkčnost celé hry co se týče herních principů a interakce s hráčem, přehrávání zvuků, rozvláknění atd.

Projekt s názvem **MadBobo-android** obsahuje vše potřebné pro běh na androidu, a to včetně připojitelných služeb google jako jsou žebříčky nebo například také nákupy za reálné peníze. V tomto projektu jsou také fyzicky uloženy veškeré textury, zvuky a jiné externí soubory, protože to vyžaduje libGDX.

Projekt s názvem **MadBobo-desktop** obsahuje vše potřebné pro běh hry na počítači s operačním systémem podporujícím prostředí Java.

Projekt s názvem **MadBobo-server** obsahuje třídy zajišťující síťovou hru více hráčů v prostředí Internet pomocí serverové aplikace.

2.3 Popis nejdůležitějších tříd

2.3.1 Třída Bobo

Tato třída představuje samotného hlavního hrdinu. Je zde uložena jeho jediná instance (což je zajištěno návrhovým vzorem jedináček, viz sekce 2.13.2).

Načítá se zde skeleton hlavního hrdiny se všemi jeho animacemi, a také se zde tyto animace ve specifických událostech (náraz do země, do nepřítele atd.) mění za jiné.

Mezi nejdůležitější metody v této třídě patří **enemyHit**, která řeší, co všechno se má stát, když hrdina přímo koliduje s nepřítelem. Nepřátel je totiž několik druhů, a při kolizi s každým z nich, se musí stát něco úplně jiného. Někteří z nich jsou speciální nepřátelé, kteří spouští minihry.

Spouštění miniher je celkem komplikované, jelikož zde dochází k různému napojování koster na jiné kostry, na což ještě není stále vyvíjené prostředí Spine dostatečně připraveno. Například když hrdina trefí speciálního nepřítele, jedoucího na dinosaurovi, shodí ho z něj a na dinosaura nasedne sám. Oba aktéři poté musí být spojeni v jednu kostru s tím, že si zachovají vlastní animace. Jediný způsob jak kostry spojit je vzít základní (root) kost kostry, kterou chceme připojit a slot některé z kostí kostry, na níž chceme připojovat. Poté se tato základní kost dá připojit do tohoto slotu stejným způsobem, jakým se tam nahrává obyčejný grafický prvek. Ve slotu je však aktivní vždy pouze jeden, což by při použití některého z existujících slotů znamenalo, že se v něm aktivní grafika vymění za připojenou kostru - grafika původní kostry by tak byla neúplná. Tahle situace se zatím nedá řešit jinak, než přidáním další kosti se slotem, který bude sloužit pouze pro účely napojení druhé kostry a jinak v něm nic nebude.

```
public void enemyHit (Hulu enemy) {
 ...
EnemyType type = enemy.getType();
Table middle = gameScreen.getMiddle();
 ...
if (gameScreen.getClickType() == ClickType.SMASH || gameScreen.getClickType() == ClickType.
    AFTER_SPECIAL) {
  switch (type) {
    case NORMAL_BOMB: {
      //napojeni koster, pricitani do statistik , zobrazeni indikatoru minihry, nastaveni
          clickType, nastaveni rychlosti apod.
    }
    case RAIDER: {
      //napojeni koster, pricitani do statistik , zobrazeni indikatoru minihry, nastaveni
          clickType, nastaveni rychlosti apod.
    }
     ...
  }
   ...
}
```

Výpis 1: Metoda enemyHit

Tato metoda také nastavuje velmi důležitý **clickType**, který určuje styl hraní spuštěné minihry, co se týče uživatelské interakce, protože některé minihry vyžadují co nejrychlejší klikání, některé zase dlouhý klik apod. O následné zpracování těchto typů interakcí a gest se stará metoda **enemyClick**.

Minihra je vždy omezena časem a končí explozí, která má sílu úměrnou úspěšností hráče v jejím hraní. Pokud se hráči daří, exploze bude velká a bude znamenat podstatné zvýšení rychlosti. Výpočet rychlosti probíhá v metodě **enemyExplode**, která se taktéž stará o vliv exploze na nepřátele voláním další metody **explode**.

```
public void explosion (float range) {
    float x = 0, y = 0;

    if (enemies != null) {
        for (int i = 0; i < enemies.size(); i++) {
            Hulu e = enemies.get(i);
            x = Math.abs(bobo.getX() − e.getX());
            y = Math.abs(bobo.getY() − e.getY());

            if (Math.sqrt(x ∗ x + y ∗ y) < range && ... // vedlejsi podminky ... && !e.isDead()) {
                e.death();
                enemyRicochet(e);
            }
        }
    }
}
```

Výpis 2: Metoda explode

Třída navíc obsahuje také odrazy hlavního hrdiny o zem, detekci kolize (viz sekce kolize 2.4.1), výpočet počátečního odpalu a menší grafické efekty, jako jsou částicové systémy, nebo také jemnou vibraci hrdiny, která nabývá na síle v závislosti na horizontální rychlosti, což velmi pomáhá navodit dojem, že hrdina světem opravdu sviští.

2.3.2 Třída Hulu

Třída hulu se stará zejména o vytvoření a nakonfigurování nepřátel pro následné narození nepřátel (více v sekci Generování nepřátel 2.6). Po zemi běžícím nepřátelům také náhodně generuje čepice, oblečení, zbraně a barvu pleti pro větší rozmanitost v metodách **setRandomHat**, **setRandomSkirt**, **setRandomWeapon** a **setRandomBody** (v pořadí).

Druhů nepřátel je více, a je tak mezi nimi při vytváření nutné vybrat. Výběr je náhodný a hodnoty šancí jsou sepsány mimo - ve třídě **EnemyType**.

Dále tato třída říká, co se má stát při omračení nepřítele hlavním hrdinou pomocí metody **death**. Po omračení se nepříteli vynuluje rychlost a nastaví animace, při které padá na zem. Vygeneruje se náhodný obnos herní měny z intervalu, který je následně ovlivněn vylepšeními a herními výhodami, aktualizují se veškeré příslušné statistiky a popřípadě také mise.

2.3.3 Třída GameScreen

Nejobsáhlejší třída celého projektu. Zpracovává a zobrazuje vše, co je viditelné ve hře. Při zobrazování herní obrazovky (kterou má tato třída celou na starost) se nejprve vykreslí obloha, hory a další prvky pozadí, např. dynamické mraky. Dále se vytvoří scéna (více o scéně v sekci Tvorba grafického rozhraní 2.5), do které se následně přidají aktivní prvky.

Herní obrazovka obsahuje mnoho prvků grafického uživatelského rozhraní, jako jsou ukazatele rychlosti, odpalu a vzdálenosti, tlačítka apod. Všechny tyto prvky jsou vytvořeny a rozmístěny v metodě **init** už při načítání této obrazovky. Zároveň se zde načtou všechny potřebné kostry a hlavnímu hrdinovi, v tuto chvíli sedícím na katapultu, se náhodně nastaví jedna z několika animací.

Třída obsahuje hlavní vykreslovací (**render**) funkci. Ta vždy nejprve aktualizuje pozice a rychlosti všem aktivním objektům, zavolá funkce pro kontrolu kolize na ty, které jsou kolizní a poté je všechny vykreslí na obrazovku. Funkce pracuje s časovou **deltou**, která udává čas mezi dvěma vykreslenými snímky v sekundách. Je zde proto, že ne na všech zařízeních hra běží se stejným počtem snímků za sekundu, což by bez delty znamenalo, že na výkonnějším zařízení bude hra oproti slabším zařízením zrychlena. Čím méně je snímků za sekundu, tím je delta větší, a tím více se objekty při dalším vykreslení posunou, jelikož jsou jí všechny vynásobeny.

Samotný **výpočet kolizí** pro veškeré objekty ve hře taktéž probíhá v této třídě (více o kolizích v sekci Kolize 2.4.1) a spadá zde i náhodné generování nepřátel (více v sekci Generování nepřátel 2.6).

Kamera je další z věcí, kterou tato třída ovládá. Je závislá na pozici hlavního hrdiny, kterého vždy začne pronásledovat, jakmile se hrdina přiblíží na určenou vzdálenost k jednomu z okrajů obrazovky. Zde se také provádí např. efekt třesení kamery při nárazu.

Pro zpracování **miniher** je zde metoda **enemyTypeSolver**. Jelikož je každá minihra jiná a má tak i jiné ovládání, je třeba zvlášť popsat, co se vlastně má dít, když je některá z nich spuštěna. Tato metoda je velmi všestranná. Počítá uplynulý čas minihry (jelikož minihry jsou časově omezené), zobrazuje speciální předměty, jež se objevují jen během miniher a hráč je má za úkol sbírat, oznamuje instanci hlavního hrdinovy (třídě bobo) že bylo kliknuto na displej resp. je prováděno gesto atd.

```
public void enemyTypeSolver (float delta) {
    switch (clickType)  {
        case ROCKET:
            //co se ma stat kdyz je  spustena minihra "raketa"
            //zpracovani ovladani (drzeni  prstu  na  displeji )
        case PLANE:
            //co se ma stat kdyz je  spustena minihra "letadlo"
            //zpracovani ovladani (drzeni  prstu  na  displeji )
            //vykresleni  cile ,  ktery  by mel hrac letadlem  trefit
            ...
```

Výpis 3: Metoda enemyTypeSolver

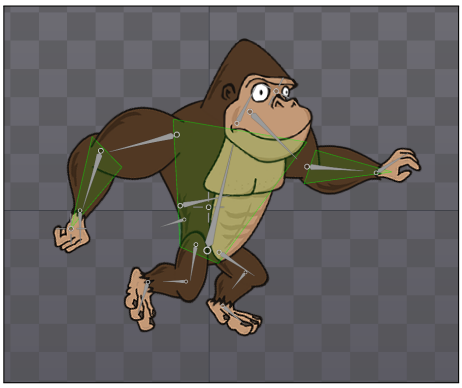
2.4 Fyzika hry

Chování, co se týče napodobení fyzikálních jevů, je vytvořeno od základu, protože k tomu určené rámce jsou v naprosté většině pro účely této hry příliš obsáhlé a komplexní. Vzhledem k tomu, že hlavní hrdina i nepřátelé mají své vlastní animace a pózy v průběhu celé hry, není zde potřeba řešit jejich dynamické kolize (například že hrdina zavadí částí svého těla o překážku a podle toho se odrazí, začne rotovat atd.). Na druhou stranu je zase nutné uchovávat větší množství atributů týkajících se např. odrazivosti, koeficientem zpomalení při dopadu apod.

Zásadní vliv zde mají herní vylepšení, které chování hlavního hrdiny ovlivňují téměř ve všech směrech (hra je na tom založena). Hrdina má svou **X** a **Y** rychlost, od které se odvozují jeho další vlastnosti. Rychlosti se přepočítávají vždy při událostech jako je dopad na zem, kolize s nepřáteli nebo překážkami, při hraní miniher apod.

2.4.1 Kolize

Kolize objektů je řešena pomocí kolizních oblastí (v angličtině bounding box), které jsou již nadefinované z prostředí Spine. Jsou dány body, které tvoří uzavřené polygony. Zde je nutné dbát na dodržení rozumného počtu bodů, jelikož s každým bodem je výpočet kolize kolizních oblastí složitější, náročnější na procesor a má poměrně velký dopad na celkovou plynulost hry zvláště v případech, kdy je na obrazovce mnoho aktivních objektů (více v sekci Optimalizace 2.12. Kolizní oblasti jsou přímo připojené k vybraným kostem, a tak kopírují pohyby i animovaných koster.



Obrázek 7: Kolizní oblast

```
public boolean intersectsWith (SkeletonWrapper skel) {
boolean intersects = false;
if (skel instanceof Hulu) {
    Hulu assoc = ((Hulu)skel).getAssociate();
    if (assoc != null) intersects = super.intersectsWith(assoc);
    }
    return super.intersectsWith(skel) || intersects;
}

...

public boolean intersectsWith (SkeletonWrapper skel) {
    if (! visible || !skel. isVisible () ) return false;

    return bounds.aabbIntersectsSkeleton(skel.getBounds());
}
```

Výpis 4: Kolize objektů

Nikdy není potřeba počítat kolizi pro všechny objekty. Nastávají situace, kdy je jich na scéně velké množství a docházelo by tak k velkým a zbytečným ztrátám na výkonu. Je totiž možné předpokládat, se kterými objekty bude v následující chvíli hlavní hrdina v kontaktu a se kterými ještě (anebo vůbec) kolidovat nebude.

Základní rozhodování je velmi jednoduché, protože spočívá pouze v kontrolování vzdálenosti objektu od hrdiny na X a Y souřadnici. Je-li například X souřadnice objektu větší, než X souřadnice hrdiny plus určená vzdálenost, tak je jasné, že ke kolizi v nejbližších momentech určitě nedojde, jelikož vzdálenost mezi objekty je ještě příliš velká. To samozřejmě platí i pokud je X menší než X souřadnice hrdiny mínus určenná vzdálenost a úplně stejný postup je i při porovnání s osou Y.

Pokud je však objekt v této určené vzdálenosti od hrdiny, bude s ním nejspíše kolidovat a je tedy nutné začít kolizi řešit přesněji. Využita je zde technika **AABB**, která slouží k rychlému zjištění kolize s objektem. Přesná polygonová kolizní oblast se zabalí do obdélníkové či čtvercové konvexní obálky a nejprve se tedy určuje, zda se protínají ony. Pokdy ano, přechází se k výpočtu přesné kolize dané zmiňovanými polygonovými oblastmi. Nejčastěji se proto počítá pouze s obálkami a jen v nejnutnějších případech, kdy jsou dva objekty kriticky blízko, proběhne výpočet definovaných kolizních oblastí.

Hra je svižná a často tak nastává situace, kdy hrdina koliduje s více dalšími objekty zároveň. Tomu jsem však kvůli hratelnosti a splnění mé vize o této hře zabránil, jelikož někdy je zkrátka vhodné upřednostnit kolizi s jedním z objektů. V těchto případech se všechny tyto objekty projdou cyklem a díky prioritě nastavené u každého z nich se zjistí, s čím bude hrdina kolidovat. Nastane však i situace, kdy hrdina koliduje s více objekty, které mají také stejnou prioritu. Zde se dále určuje, ke kterému z objektů se stejnou prioritou je hrdina blíže (se kterým koliduje více).

2.4.2 Výpočet odpalu

Každá hra začíná odpalem z katapultu, jehož síla závisí na tom, jak hráč trefí odpalovací metr a na jakou úroveň má vylepšený katapult. Určuje se zde počáteční rychlost letu. Hodnota proměnné **strokeTime** je určená časem, který má počátek vždy v perfektním odpalu a přičítá se (čím je strokeTime menší, tím lépe hráč odpal trefil).

```
public void stroke () {
 ...
speedX = catapultSpeedX / strokeTime;
speedY = catapultSpeedY / strokeTime;
setVelocity(speedX * Upgrades.getCatapult(), speedY * (1 + Upgrades.getCatapultLevel() *
    (Upgrades.catapultMultiplier / 2)));
 ...
}
```

Výpis 5: Výpočet odpalu

2.5 Tvorba grafického rozhraní

Pro tvorbu grafického uživatelského rozhraní je v rámci libGDX systém **Scene2D**[8][9], který velmi urychluje práci, co se týče návrhu rozložení komponent. Chová se jako **tabulka**, jejíž každé buňce je možné zvlášť nastavit zarovnání, velikost, roztahování apod. Podporuje také spuštění v režimu debug, které graficky (pomocí barevných rámečků) znázorní buňky, pro lepší přehled o jejich chování, což je velmi užitečné.

2.5.1 Scéna, tabulka a objekt

Základním prvkem je **scéna** (anglicky stage), která zpracovává vnější vstupní signály (doteky a gesta) a spouští je na příslušných prvcích rozhraní. Scéna musí mít vždy nastavený výřez (anglicky viewport), který si lze představit jako kameru, určující jak bude obsah zobrazen na obrazovce (viz sekce Různá rozlišení 2.5.2). V projektu je využito hned několik scén. Abych nemusel konfigurovat každou zvlášť, vytvořil jsem pomocnou abstraktní třídu **stageScreen**, která se postará alespoň o to nejzákladnější nastavení.

```
public abstract class StageScreen implements Screen {
 ...
public StageScreen () {
    stage = new Stage() {
        @Override
        public boolean keyDown (int keyCode) {
            if  (keyCode == Keys.ESCAPE || keyCode == Keys.BACK) Dialogs.showExitDialog(stage,
                screen);
            return false;
        }
    };
    screen = this;
}
```

Výpis 6: Tvorba scény

Do vytvořené scény se pak přidá hlavní **tabulka** (anglicky table), která bude sloužit jako rodič všem dalším objektů do ní vložených podobně jako u HTML. Prvky grafického rozhraní jako např. tlačítka, ukazatele průběhu apod. samy o sobě nemají nastavenou velikost ani pozici. To vše totiž vždy určuje jejich nadřazený rodič. Lze jim však nastavit minimální a maximální velikosti, na které bude rodič brát při vykreslení ohled.

```
table  = new Table();
table.setName("MAIN_TABLE");
table.setBackground(Utils.getColorDrawable(Utils.getColorFromRGB(185, 138, 83, 1f)));
 ...
```

Výpis 7: Tvorba tabulky

Objekty se do tabulky přidávají jednoduše pomocí příkazu **tabulka.add(objekt)**. Vkládají se za sebe směrem doprava. Pro vytvoření nového řádku v tabulce slouží **tabulka.row()**. Objektům je nejprve nutné vytvořit **styl**. Styly jsou uložené ve třídě Assets.

```
//vytvoreni a nastaveni stylu
LabelStyle lblText = new LabelStyle();
lblText.font = Assets.get(Fonts.WHITEBROWN36);

//vytvoreni textove komponenty typu label
Label text = new Label(lang.get("text"), lblText);
table.add(or).padTop(-10);
table.row();
```

Výpis 8: Tvorba objektu

Vlastnosti buněk tabulky:

- **Rozšíření** (Expand) - zabere místo okolo objektu ve velikosti buňky
- **Zarovnání** (Alignment) - zarovnání objektu v buňce na stranu, doprostřed apod.
- **Vyplnění** (Fill) - Vyplní rodičovskou buňku daným objektem
- **Velikost** (Widget size) - nastavení minimální a maximální velikosti objektu
- **Vnitřní odsazení** (Padding) - místo navíc okolo objektu
- **Sloučení sloupců** (Colspan) - sloučení sousedních sloupců do jednoho
- **Rovnoměrnost** (Uniform) - dá stejnou velikost všem buňkám, na které je použita

Tyto vlastnosti lze libovolně skládat za sebe a je tak možné jednoduše umístit a nastavit objekt či buňku co se pozice, velikosti a chování v tabulce týče jakkoliv je potřeba. Vlastnosti se nastavují zároveň s přidáváním objektu do tabulky.

```
table.add(top).expandX().fillX().padTop(20);
...
table.add(smashes).expandY().bottom().padBottom(10).padLeft(50);
...
table.add(bottom).bottom().fillX().expandX().colspan(4);
```

Výpis 9: Skládání vlastností

Komponent je velké množství a dají se velmi dobře a snadno upravovat. V projektu jsem kromě těch základních, jako jsou tlačítka, popisky, obrázky apod., využil i komplexnější komponenty **stack** a **scrollPane**.

Stack je komponenta, která umožňuje vložení několika objektů přes sebe s tím, že každý z objektů dědí rodičovské vlastnosti stejně, jako je tomu u obyčejné buňky. To je užitečné při tvorbě složených komponent, kde je třeba mít v buňce více než jeden objekt.

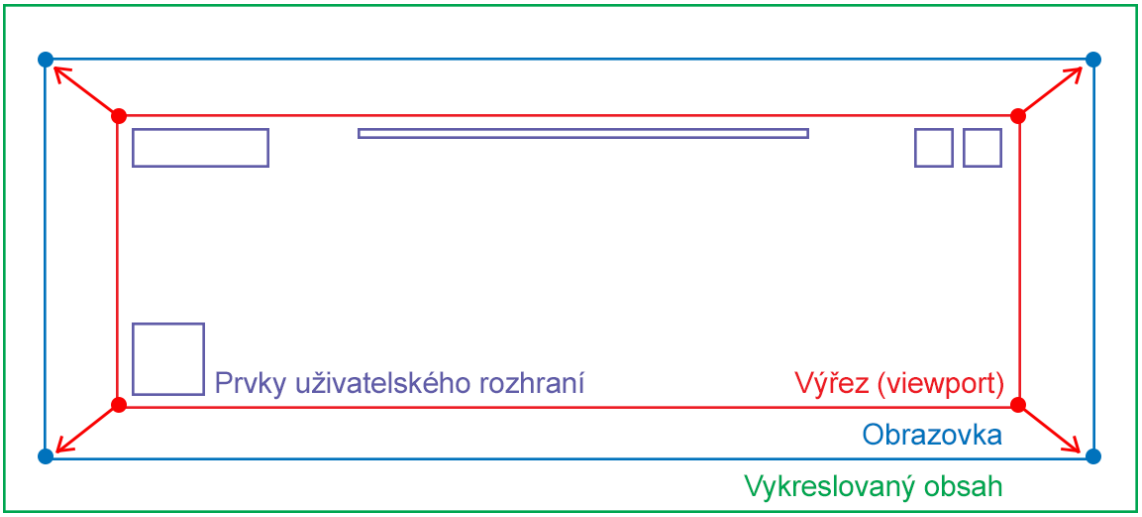
V projektu je stack využit například u ukazatele průběhu aktuálního kola, kdy na pozadí je obrázek znázorňující délku světa a v popředí šipka ukazující kde se hrdina nachází.

ScrollPane je komponenta, která umožňuje tzv. **scroll** - tedy posouvání svého obsahu (podřízených objektů). Pokud je v této komponentě více obsahu, než dokáže zobrazit staticky, objeví se na příslušné straně posuvníky, pro znázornění pokračujícího obsahu. V základu se však komponenta roztahuje na celý svůj obsah, ať už je jakkoliv velký, proto je nutné omezit jeho velikost nastavením maximální velikosti alespoň v jednom z rozměrů. Využil jsem ji například na obrazovce pro nákup vylepšení a herních výhod v záložce "Goods", kde by se obsah zkrátka celý najednou nevešel.

2.5.2 Různá rozlišení

[5] Zásadním problémem při vývoji her či aplikací pro mobilní zařízení, je jejich rozlišení. Existuje nepřeberné množství mobilních telefonů a tabletů, které se rozlišením liší. Lze sice nastavit pevné rozlišení a použít ho pro všechna tato zařízení, výsledek ale nebude na většině z nich vypadat dobře anebo bude dokonce různě zdeformován (roztáhnut nebo smrštěn horizontálně či vertikálně), kvůli různému poměru stran.

V projektu je problém vyřešen tak, že je prvně určena **plocha vykreslovaného obsahu**. Jedná se o obdélník pevného rozlišení, který je vyplněn grafickými prvky pozadí. Tento obdélník vždy částí přesahuje velikost obrazovky zařízení, což zajišťuje rezervu pro zařízení s jiným poměrem stran a nedochází tak k deformaci. **Plocha výřezu**, kterou udává virtuální rozlišení, se poté přizpůsobí reálnému rozlišení obrazovky. Po přizpůsobení se do scény vloží grafické prvky popředí (tlačítka a ukazatele na obrazovce), které jsou různě ukotvené např. k jednomu z rohů výřezu, ke středu horní strany výřezu apod.



Obrázek 8: Zpracování pro různá rozlišení

2.6 Generování nepřátel

Nepřátelé jsou náhodně generováni s tím, že každý z nich má na vygenerování předem určenou šanci v třídě typu enum s názvem **enemyType**. Někteří nepřátelé mají šanci konstantní, u některých je ovlivněna vylepšeními a použitými herními výhodami. Typy nepřátel jsou navíc dynamicky načítány a tak pro přidání dalšího stačí připsat novou hodnotu do třídy enemyType a do složky s kostrami vložit stejnojmennou kostru.

Jako první se náhodně vybere, jaký typ nepřítele bude generován, od čehož se může odvíjet jeho následná výška narození a horizontální rychlost. Někteří z nepřátel jsou speciální, ti mají jiné chování než všichni ostatní nepřátelé a zároveň při kolizi s nimi dojde ke spuštění minihry (více o minihrách v popisu třídy bobo.java 2.3.1).

Dále je třeba určit, na jaké **X** (horizontální) pozici se má nepřítel vygenerovat. Ta je dána aktuální polohou kamery vzhledem k světu, jelikož nepřítel se musí vygenerovat za jejím okrajem dostatečně daleko na to, aby nebylo vidět jeho narození a dostatečně blízko na to, aby jich za okrajem kamery nebylo zbytečně moc.

Výpočet druhé souřadnice **Y** závisí na typu nepřítele. Část z nich se generuje na zemi a část v určeném výškovém rozsahu. Každý nepřítel má předem určeno, v jakém rozmezí výšky se může narodit.

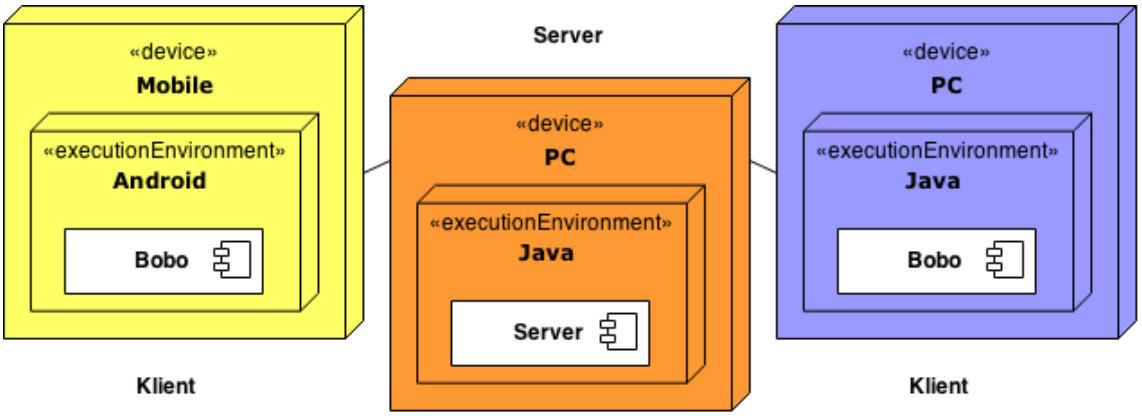
Posledním krokem je samotné vytvoření instance tohoto nepřítele a nastavení jeho horizontální rychlosti, která se vždy odvíjí od rychlosti hlavního hrdiny s přidanou náhodnou složkou. Tato složka je záporná, jelikož nepřátelé se vždy generují za pravým okrajem obrazovky a hrdina je poté dožene.

```
public void addEnemy () {
Hulu enemy;

type = EnemyType.getRandom((bobo.getGatesHit() + 1) * 2, bobo.getSpecialChanceMultiplier());

if  (!(( gateLeftLeft.rect.x-(gameViewport.getCamera().position.x+VIRTUAL_WIDTH) < 2000) &&
     (gateLeftLeft.rect.x-(gameViewport.getCamera().position.x+VIRTUAL_WIDTH) > -1000)) {

   float posX = cameraPosition.x + VIRTUAL_WIDTH + Utils.randomRange(200,
        VIRTUAL_WIDTH);
   float posY;

   if (type == EnemyType.FLYING) {
      posY = Utils.randomRange(VIRTUAL_HEIGHT * 1.5f, heaven.getTop() -heaven.getHeight());
   } else if (type == EnemyType.TICKET) {
      posY = Utils.randomRange(VIRTUAL_HEIGHT * 1.0f, heaven.getTop() -heaven.getHeight());
    ...
   enemy = new Hulu(type, posX, posY, this);
   enemy.setVelocityX(Utils.randomRange(bobo.getVelocityX() -2400, bobo.getVelocityX() + 360));

   enemies.add(enemy);
}
```

Výpis 10: Generování nepřátel

2.7 Síťová hra

[3] Hra více hráčů přes internet funguje na principu serveru v roli prostředníka. Klienti zadají IP adresu a připojí se na server, který vždy čeká, než se připojí oba dva. Server poté zjistí informace o obou klientech a přepošle je tak, aby oba klienti věděli vše potřebné o klientovi protějším. Poté klienti posílají pouze nejnutnější zprávy o pozici svého hlavního hrdiny a stavu hry na server, který je opět přijímá a přeposílá protějšímu klientovi.



Obrázek 9: Diagram nasazení.

Projekt je, co se týče síťové hry rozdělen na tři části - klientská, serverová a společná část. Klientská část a serverová část jsou jinak odděleny, ale obsahují balík se stejným názvem ("bobo"), který se díky tomu při překladu sloučí do jednoho - společná část.

Hráč může při hře vidět na pozadí ducha protihráče. Jeho pozici má možnost sledovat na ukazateli průběhu kola. Žlutá šipka náleží hráči, červená šipka protihráči.

Postup komunikace:

1. Hráč zadá přezdívku a IP adresu serveru.
2. Klient vytvoří tzv. **socket** a pokusí se připojit na server, který již na připojení čeká.
3. Server čeká na připojení dvou hráčů.
4. Server vytvoří tzv. **repeater**, který odstartuje dvě vlákna, jedno pro každého hráče.
5. Vlákno naslouchá a čeká, až mu od hráče přijde nějaká zpráva.
6. Hráč odešle zprávu (např. o své pozici).
7. Vlákno ji po přijetí odešle druhému hráči.
8. Do žurnálu (log) serveru se zapíše, o jakou zprávu šlo.

Socket [10] si lze představit jako koncový bod obousměrného připojení dvou aplikací přes síť. Klientův socket komunikuje se socketem serveru přes protokol Net, který mimo jiné zajišťuje také kompatibilitu síťového propojení zařízení běžícími na různých platformách. Obsahuje vstupní a výstupní tok dat pro odesílání a přijímání zpráv.

2.8 Přehrávání zvuku

Pro přehrávání zvuku ve hře má libGDX vlastní systém [6]. Předně je zvuk rozdělen na zvukové efekty a hudbu. Tento rámec podporuje přehrávání zvukových souborů ve formátu ogg, mp3 a wav. Osobně jsem využil formát ogg, který má trochu menší velikost než mp3 nebo wav a na kvalitě to téměř není znát.

2.8.1 Zvukové efekty

Zvukové efekty jsou krátké zvukové sekvence, které mají většinou jen pár sekund a jsou přehrávány, když ve hře dojde k nějaké specifické události. Například pokud dojde ke kolizi hlavního hrdiny s nepřítelem, přehraje se zvuk úderu apod. Tyto zvuky nesmí mít velikost větší než 1mb a načítají se celé přímo do paměti RAM. Díky tomu jsou přístupné okamžitě a lze je bez problému přehrávat i ve vysoké rychlosti za sebou. Pro zvukové efekty je zde rozhraní Sound.

```
//vytvoreni instance zvuku a nacteni souboru
private Sound sound;
...
sound = Gdx.audio.newSound(file);

//prehrani zvukoveho efektu
Assets.get(Sounds.STARTUP).play();

//odstraneni zvuku z pameti
Assets.get(Sounds.STARTUP).dispose();
```

Výpis 11: Zvukové efekty

Načteným zvukům lze upravovat hlasitost, výšku apod. Například některé z nich byly hlasitější než jiné, a tak jsou ztišeny přímo v kódu metodou **setVolume();**

setVolume je metoda určená ke změně hlasitosti jednotlivých instancí zvuku. Hodnoty hlasitosti jsou v rozmezí od 0 do 1, kde 0 znamená ticho a 1 maximální hlasitost.

setPitch nastavuje výšku/rychlost instance zvuku. Základní hodnota je 1, nižší hodnoty znamenají zpomalení přehrávání zvuku (čímž bude zvuk hlubší), vyšší hodnoty jej naopak zrychlí (čímž bude zvuk tenčí). Hodnoty však musí být mezi 0,5 a 2,0.

Ve hře je vytvořen systém postupné změny výšky hlasitosti, když hráč sbírá herní měnu. Při sebrání se vždy výška zvuku nepatrně navýší, což znamená, že čím více hráč sbírá, tím je tento zvuk vyšší. Pokud hráč po určitou dobu nesebere nic, výška zvuku se vrátí na svou základní hodnotu 1.

```
float pitch = 1.5; float volume = 0.5;
bananaCollectSound.setPitch(bananaCollectSound.play(), pitch);
bananaCollectSound.setVolume(bananaCollectSound.play(), volume);
```

Výpis 12: Úprava zvuků

setLooping vytváří zvukovou smyčku. Zvuk ve smyčce se vždy po skončení začne ihned přehrávat znovu. Nabývá hodnot true nebo false - opakovat nebo neopakovat.

setPanning dokáže "naklonit"zvuk na jednu stranu sterea. Hodnota 0 znamená, že se bude zvuk přehrávat vyváženě na levém i pravém zvukovém kanále. Hodnoty od 0 do -1 budou zvuk zesilovat na levém kanále a ztišovat na pravém, hodnoty od 0 do jedné je naopak zesilovat na pravém a ztišovat na levém kanále. Díky této metodě se dá docílit prostorového zvuku.

2.8.2 Hudba

Soubory s hudbou mají mnohem větší velikost než ty se zvukovými efekty, a tak není možné je načítat přímo do paměti RAM. Místo toho se vytvoří datový proud, který postupně čte zvukový soubor z disku a rovnou jej přehrává.

Hudbě lze nastavit hlasitost, výšku a vyvážení zvuku stejně jako zvukovým efektům. Jsou zde navíc funkce **isPlaying()**, která zjišťuje, zda daná hudba právě hraje, **isLooping()**, pro zjištění, jestli hraje ve smyčce a **getPosition()**, jenž vrací pozici právě přehrávané části skladby v sekundách.

```
MusicWrapper music = Assets.get(Music.ANT4505_STRINGS_BASS);
music.setLooping(true);
music.play();
```

Výpis 13: Hudba

2.9 Statistiky

Ve hře se zaznamenává téměř vše, co hráč udělal (kolik utratil, kolik metrů uletěl v posledním kole nebo celkově atd.). Hodnoty se uchovávají a používají zejména ke kontrole plnění misí a herních úspěchů. Statistiky se spravují ve třídě **Statistics** a jsou deklarovány ve třídě **StatisticsData**.

Kvůli bezpečnosti se statistiky při ukládání vždy zašifrují systémem **base64**. To znemožní smysluplnou editaci souboru, ve kterém jsou uloženy, a tak i podvádění.

Ukládání statistik probíhá vždy, když hráč provede specifickou akci, jako je například nákup v obchodě, výhra v kole štěstí a podobně. Problém nastává, když je hráč přímo ve hře (ovládá hlavního hrdinu), jelikož zde probíhá velká spousta akcí, které je nutné ukládat. Kdyby se měly statistiky aktualizovat a ukládat ihned, bude to zbytečně náročné na výkon, jelikož by se vždy musel načíst soubor, rozšifrovat data, aktualizovat je, zašifrovat zpět a uložit. Proto se přímo ve hře statistiky ukládají co 2 sekundy, což mi přišlo akorát. Tímto je také zajištěno, že hráč neztratí nic z toho, co v daném kole nahrál, dojde-li například k nechtěnému přerušení aplikace během hraní.

2.10 Vícejazyčnost

Hra je lokalizovaná do češtiny a angličtiny s tím, že se sama pokouší nastavit jazyk podle systémového jazyku zařízení, s čímž velmi pomáhá rámec libGDX. Jazyk jde také změnit později v aplikaci a to přímo za běhu.

Nejprve je nutné vytvořit soubory zvané **properties**, jeden pro každý jazyk. Vždy by měl existovat výchozí soubor properties, který se použije v případě, že aplikace nerozpozná systémový jazyk zařízení nebo tento jazyk není aplikací podporován. Výchozí i přídavné soubory properties mají stejný základní název. V mém případě je výchozí soubor pojmenován **lang.properties** a přídavný soubor s překladem do češtiny **lang_cs.properties**.

Lang_cs.properties

```
noThanks=Né, děkuji
next=Další...
continue=Pokračovat...
go=GO!
back=Zpět
invited=Pozváno: {0}
resume=Pokračovat
videoNotAvailable=Video ještě není\npřipraveno.
changeOutfit=Změnit ohoz
resetOutfit=Obnovit ohoz
okay=Oukej
missionsCompletedText={0} z {1} misí splněno.
missionsOf={0}\nz\n{1}
```

Lang.properties

```
noThanks=No, thanks
next=Next...
continue=Continue...
go=GO!
back=Back
invited=Invited: {0}
resume=Resume
videoNotAvailable=Video is not yet\nready to play.
changeOutfit=Change outfit
resetOutfit=Reset outfit
okay=Okay
missionsCompletedText={0} of {1} missions completed.
missionsOf={0}\nof\n{1}
```

Obrázek 10: Soubory properties

Obsahem těchto souborů je vždy **klíčová hodnota**, která slouží jako identifikátor a kní připojený odpovídající text v daném jazyce. V textu mohou být obsaženy i argumenty, které se píšou do složených závorek a jsou to vždy čísla od 0. Za tyto argumenty se poté dají dosadit například číselné hodnoty.

Dále je potřeba samotná instance třídy **I18Bundle**, která slouží k uchování a přeposílávání lokalizovaných řetězců ze souborů. Při vytváření této instance se zároveň definuje cesta k souborům properties.

K samotnému načtení chtěného textu slouží metoda **format**, která požaduje klíčovou hodnotu a popřípadě také potřebné argumenty definované v souborech properties.

```
// vytvoreni I18NBundle s deklaraci cesty k souborum properties
MainClass.lang = I18NBundle.createBundle(Gdx.files.internal(MainClass.dataPath + "i18n/lang"));

// nacteni textu pomoci metody format
textPrice = lang.format("buyFor", Utils.separateThousands(price));
```

Výpis 14: Práce s více jazyky

2.11 Práce se kostrami

Pro jeden animovaný objekt jsou zde vždy 3 soubory nehledě na to, kolik má daný objekt animací, kostí apod. Tyto 3 soubory jsou vygenerovány programem Spine, ve kterém jsem tvořil herní animace.

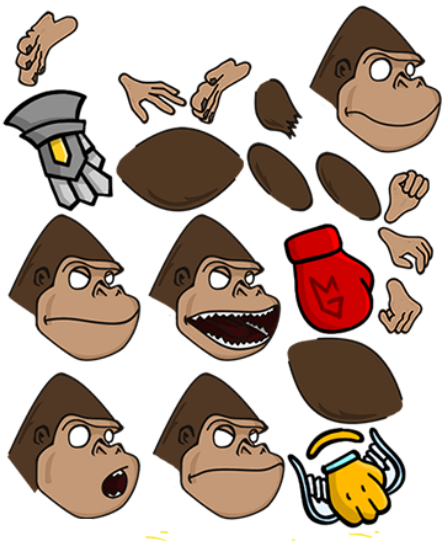
Atlas

```
bobo.png
size: 1022,1018
format: RGBA8888
filter: Linear,Linear
repeat: none
SODA
  rotate: false
  xy: 156, 300
  size: 52, 92
  orig: 52, 92
  offset: 0, 0
  index: -1
arm_left
  rotate: true
  xy: 307, 434
  size: 82, 124
  orig: 82, 130
  offset: 0, 2
  index: -1
arm_right
  rotate: true
        ...
```

JSON

```
{"skeleton":{"spine":"1.9.17","h
ash":"HhTSBpAlPLk0sIdatMAkQewxqJ
I","width":406.82,"height":380.0
5},"bones":[{"name":"root","y":3
.79},{"name":"bone17","parent":"
root","length":57.89,"x":-32.06,
"y":1.73,"rotation":11.88},{"nam
e":"bone18","parent":"root","len
gth":28.89,"x":-27.5,"y":-13.91,
"rotation":-162.8},{"name":"bone
","parent":"bone17","length":42,
"x":8.3,"y":-46.66,"rotation":-1
10.93},{"name":"bone4","parent":
"bone17","length":48.38,"x":32.2
,"y":-60.47,"rotation":-45.7},{"
name":"bone7","parent":"bone17",
"length":123.74,"x":19.44,"y":-5
5.82,"rotation":64.34},{"name":"
bone11","parent":"bone7","length
":76.3,"x":163.77,"y":-9.61,"rot
ation":-121.33},{"name":"bone14"
,"parent":"bone7","length":48.5,
"x":147.64,"y":1.6,"rotation":-1
        ...
```

Textura



Obrázek 11: Náhled souborů k animaci.

- **Atlas** - V hlavičce tohoto souboru je nastavena cesta k textuře, její velikost, filtrování, formát a opakovatelnost. Tělo souboru tvoří vždy název jedné z částí textury (např. "hlava") a specifikace jejího umístění ve společné textuře tzn. její pozice, velikost, posun, rotace apod. Dá se říct, že atlas každou část textury zabalí do pomyslné obálky, díky čemuž lze načítat veškeré obrázky potřebné k animaci z jednoho fyzického obrázkového souboru. Šetří se tím paměť na grafickém akcelerátoru a snižuje se také velikost souboru a tím i celé hry.
- **JSON** - Soubory JSON [4] jako takové, slouží především ke snadné výměně dat mezi aplikacemi. V mém případě jsou v tomto souboru uloženy veškeré informace o kostře animace a také o samotných animacích, klíčových snímcích, transformacích atd. Soubor je v takzvaném minimálním režimu - všechny informace jsou na jednom řádku těsně za sebou.
- **Textura** - Bitmapa obsahující veškeré grafické prvky animace. Tento soubor je taktéž vygenerován programem Spine, do kterého jsem veškeré tyto grafické prvky nahrál zvlášť a on už se postaral o jejich spojení do jedné bitmapy.

Načítání koster i s animacemi je snadné, jelikož libGDX přímo podporuje Spine. Třídy s metodami pro práci se kostrami jsou zde již hotové, a tak je stačí použít, popřípadě před použitím mírně poupravit ke svému obrazu.

K načítání slouží metoda **SkeletonData load**, která využívá třídu **AssetManager** rámce libGDX. AssetManager je správce načítaných souborů. Uchovává je na jednom místě, při načtení stejného souboru dvakrát pouze vytvoří odkaz na místo v paměti a umožňuje asynchronní nahrávání, díky čemuž lze sledovat také jeho průběh. Je velmi užitečný při práci s velkým množstvím textur, zvuků apod.

```
public SkeletonData load (AssetManager assetManager, String fileName, FileHandle file,
      SkeletonLoaderParameter parameter) {

TextureAtlas atlas = assetManager.get(file.parent() + "/"
   + file.nameWithoutExtension() + ".atlas", TextureAtlas.class);

if (file.extension().equals("json")) {
   SkeletonJson json = new SkeletonJson(atlas);
if (parameter != null) json.setScale(parameter.scale);
   return json.readSkeletonData(file);
} else {
   SkeletonBinary binary = new SkeletonBinary(atlas);
   if (parameter != null) binary.setScale(parameter.scale);
      return binary.readSkeletonData(file);
   }
}
```

Výpis 15: Načítání animace

K samotnému vykreslení na obrazovku je zde třída **SkeletonRenderer** a její metoda **draw**. Tato metoda za pomocí cyklů projde všechny součásti dané kostry (kosti, sloty a do nich přidané textury) a vykreslí je na obrazovku v pořadí, které se definuje už v prostředí Spine. Toto pořadí se nazývá jednoduše **DrawOrder** a slouží k určení, která část kostry bude v pozadí a která v popředí. Například ruce hlavního hrdiny, kde jedna ruka musí být vykreslena za jeho tělem a druhá ruka naopak před ním.

Po vytvoření instance kostry je možné snadno příkazem **setAnimation()** vybírat jeho aktuální animaci. Také lze dynamicky za běhu animace měnit veškeré textury kostry a dokonce na sebe kostry napojovat. Lze totiž načíst jednu kostru a do slotu mu přiřadit kostru další. Zde ale nastávají menší problémy s animováním jednotlivých na sebe napojených koster (napojování koster řeší třída bobo.java 2.3.1).

2.12 Optimalizace

Jelikož je hra určená také pro mobilní zařízení, je nutné ji k tomu přizpůsobit. Mobilní procesory zdaleka nedosahují výkonu procesorů určených pro stolní počítače, a tak je rozdíl v možnostech zpracování velmi velký.

Předně musel být snížen počet na obrazovce viditelných animovaných nepřátel a poletujících objektů, což už samotné znamenalo podstatný nárust počtu vykreslovaných snímků za sekundu.

Dále je zde omezení vykreslování veškerých grafických prvků na oblast mírně přesahující displej zařízení. Tyto prvky se vytvoří těsně před okrajem obrazovky než na ni vstoupí a zmizí okamžitě, až se zase celé dostanou za okraj. Díky tomuto je viditelné vždy jen minimální možné množství prvků.

Optimalizací prošly také kolizní oblasti všech předmětů, určené pro výpočet jejich kontaktu. Jak již bylo zmíněno v sekci "Fyzika hry"(viz sekce Kolize 2.4.1), kolizní oblasti jsou určené body, tvořícími uzavřené polygony. Čím více je těchto bodů, tím je výpočet pro procesor náročnější. Zmenšením jejich počtu, došlo k výraznému zlepšení plynulosti jen s minimálním dopadem na hratelnost (vzhledem ke svižnosti hry je dopad téměř neviditelný, jelikož v rychlosti, ve které se hrdina pohybuje světem, je velmi obtížné postřehnout rozdíl).

Kolize se navíc nepočítají, dokud daný objekt není na X a Y souřadnici v určené vzdálenosti od hlavního hrdiny. Je totiž možné předpokládat, se kterým objektem bude hrdina v nejbližší chvíli nejspíše kolidovat v závislosti na horizontální a vertikální vzdálenosti od něj. Například pokud je X daného objektu větší než X hlavního hrdiny + určená vzdálenost, kolize se pro něj nepočítá vůbec. Jakmile je však v tomto rozmezí, přistupuje se k výpočtu kolize metodou **AABB**. Ta spočívá v tom, že se polygonová kolizní oblast zabalí do obdélníkové konvexní obálky a pokud dojde ke kolizi s ní, pak se teprve řeší detailnější kolize využívající vícebodové polygonové kolizní oblasti (viz sekce 2.4.1).

Hra obsahuje několik vrstev pozadí, které se pohybují s rozdílnou rychlostí, pro vytvoření dojmu perspektivy. Šířka obrazovek dnešních mobilních zařízení je již velmi vysoká (mnohdy dokonce předčí šířku obrazovky monitoru stolního počítače), a tak je nemožné pokrýt celou tuto šířku jednou texturou (resp. je to možné, ale vrstev je více a textury by byly obrovské, což by mělo za následek podstatné zhoršení výkonu). Pozadí jsou proto nařezána na menší části, které se jednoduše opakují za sebou tak, aby vždy před vstupem na obrazovku byla již připravena následující textura a naopak textura, která obrazovku opustí, byla co nejdříve zničena.

2.13 Návrhové vzory

2.13.1 Command (příkaz)

K získání hodnot ze statistik, misí a herních úspěchů hráče jsem použil návrhový vzor **Command** [7]. Jedná se o jednoduché rozhraní, které obsahuje pouze metodu s příkazem execute. Návrhový vzor Command se využívá v situacích, kdy je třeba vykonání nějakého příkazu zapouzdřit do třídy a tu jako proměnnou předávat v aplikaci. Zapouzdřený příkaz by měl obsahovat pouze informace ke svému vykonání.

```
//rozhrani command
public interface CommandWithReturn<T> {
    public T execute();
}

// trida vyuzivajici rozhrani command
public class GetBoboSmashes implements CommandWithReturn<Integer> {

    @Override
    public Integer execute() {
        return Statistics.data.boboSmashesTotal;
    }
}
```

Výpis 16: Návrhový vzor Command

2.13.2 Singleton (jedináček)

Návrhový vzor jedináček slouží k zajištění toho, aby v celém programu běžela pouze jedna instance třídy. V mém případě je jedináčkem třída Bobo (tedy třída, která obsahuje veškeré informace o hlavním hrdinovi). Tento návrhový vzor zabezpečí, že třída bude mít pouze jednu instanci a bude k ní mít přístup kdokoliv (jakýkoliv jiný objekt).

3 Uživatelská dokumentace

3.1 Základní principy

Hra začíná příběhem o hlavním hrdinovi zvaném Bobo. Bobo je opice, která byla zajata příslušníky kmene Hulu a snaží se dostat z jejich spárů.

Cílem hry je dostat se na svobodu. To ale není tak jednoduché, protože na cestě čeká spousta překážek. Hráč vždy začíná na katapultu, který jej po dotyku kamkoliv na obrazovku vymrští silou odvíjející se toho, jak se mu podaří trefit odpalovací metr. Hráč poté letí vzduchem, odráží se od pozemních i vzdušných nepřátel, od země a dalších objektů. Každý náraz jej ale zpomaluje, a tak se zprvu nedostane daleko.

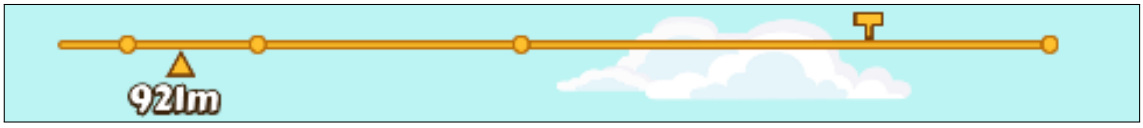
Obrázek 12: Odpal

Obrázek 13: Obrázky ze hry

Ovládání hry je jednoduché. Odražení se od nepřátel doplňuje energii na **vzdušný úder**, kterým hráč pošle hrdinu k zemi, kde odrovná nejbližší nepřátele. Použití úderu vždy stojí stejné množství energie. Celkový počet úderů se dá navýšit vylepšováním.

Hráč se s každým kolem a vylepšením má možnost dost dále. K lepší kontrole postupu hrou je zde **ukazatel průběhu**, který zobrazuje aktuální pozici vzhledem k celé mapě, vzdálenost od katapultu v metrech, pozici bran a taky velmi důležitou ikonku hráčovy největší dosažené vzdálenosti.

Rychlost je základ. K jejímu sledování slouží tachometr v levé spodní části obrazovky. Je na něm také červeně vyznačen limit maximální rychlosti, který jde zvýšit vylepšováním hlavního hrdiny.



Obrázek 14: Ukazatel průběhu

3.2 Úvodní obrazovka

Po zapnutí hry se hráč nachází na úvodní obrazovce, odkud má přístup do všech hlavních částí hry.

Obrázek 15: Úvodní obrazovka

(1) Tlačítko **Hrát** slouží ke skoku na obrazovku vylepšení a následnému spuštění hry.

(2) Tlačítko **Survival** zapíná hru typu Survival.

(3) Tlačítko **Kolo štěstí** otevírá kolo štěstí, kde může hráč vyhrát různé ceny.

(4) Tlačítko **Mise** vypisuje hráči aktuální herní mise.

(5) Tlačítko **Herní úspěchy** vypisuje hráči všechny jeho dosažené úspěchy ve hře.

(6) Tlačítko **Nastavení** zobrazí dialog nastavení chování hry.

(7) Tlačítko **Statistiky** zobrazí dialog se statistikami hráčova postupu.

(8) Tlačítko **Žebříčky** je propojené se službami Google Play a po kliknutí hráči ukáže, jak si stojí v různých ohledech hry, vzhledem k ostatním hráčům.

(9) Odpočet ve spodní části obrazovky ukazuje, za jak dlouho hráč dostane los do kola štěstí zdarma.

(10) Tlačítko **Připoj se** slouží k připojení se do komunity hry na sociálních sítích.

3.3 Obrazovka vylepšení

K tomu, aby se hráč dostal dále, je nutné hlavního hrdinu a jeho okolí vylepšovat. Vylepšení vždy kladně ovlivňují vlastnosti hrdiny jako například výšku jeho odrazů, maximální dosažitelnou rychlost, sílu odpalu katapultu, šanci na střetnutí speciálních nepřátel spouštějících minihry a další. Veškeré vylepšení stojí určitý obnos **herní měny**, kterou lze získat z nepřátel, kola štěstí a plněním misí.

Kliknutím na tlačítko Hrát na Úvodní obrazovce se hráč dostane na Obrazovku vylepšení. Zde si může vylepšit hlavního hrdinu podle vlastní strategie tak, aby letěl co nejrychleji a nejdále, získával co největší odměny atd.

Obrázek 16: Obrazovka vylepšení

(1) Tlačítkem **GO!** se hráč přesouvá na Hrací obrazovku a hra začíná.

(2) Tlačítko **Zpět** hráče vrátí zpět na Úvodní obrazovku.

(3) Záložky **Bobo, Hulu a Goods** uživateli nabízí, co všechno může ve hře vylepšovat. Jedná se o vylepšení hlavního hrdiny, ale také například šancí na střetnutí speciálních postaviček, které spouštějí minihry.

(4) Ukazatel graficky znázorňuje úroveň vylepšení vybraného typu.

(5) Tlačítko **Koupit** slouží k nákupu vybraného vylepšení.

3.4 Vylepšení

Vylepšováním vlastností hlavního hrdiny a jeho okolí, se hráč ve hře dostává dále.


Lepší úroda - Hráč získá více herní měny (banánů) z nepřátel.


Pružnost - Zvyšuje odraz od země a nepřátel.


Síla katapultu - Navýšení síly odpalu katapultu.


Ohnivý úder z nebes - Dopad z nebe na zem bude rychlejší a způsobí větší škody.


Dětský olej - Odrazy od země a nepřátel hráče tolik nezpomalí.


Soda smash - Navýšení počtu vzdušných úderů.


Tachometr - Posunutí limitu maximální rychlosti.


Kulturistika - S každou úrovní snadnější prorážení bran.


Balón - Zvýší šanci na střetnutí minihry Balón.


Dinosaurus - Zvýší šanci na střetnutí minihry Dinosaurus.


Pogo tyč - Zvýší šanci na střetnutí minihry Pogo tyč.


Plátěný štít - Zvýší šanci na střetnutí minihry Plátěný štít.


Raketa - Zvýší šanci na střetnutí minihry Raketa.


Banánovník - Zvýší šanci na střetnutí nepřátel nesoucích banány.


Bomber - Zvýší sílu nepřátel nesoucích bombu.


Letadlo - Zvýší šanci na střetnutí minihry Letadlo.


Plášť - Zvýší šanci na střetnutí minihry Plášť.


Hlídač - Sníží sílu hlídačů, kteří poté hráče při nárazu tolik nezpomalí.

K proražení bran je nutné mít odpovídající rukavici.


Chňapka - Tato rukavice je potřeba k proražení první brány.


Boxovací rukavice - Tato rukavice je potřeba k proražení druhé brány.


Rytířská rukavice - Tato rukavice je potřeba k proražení třetí brány.


Rukavice svobody - Tato rukavice je potřeba k proražení poslední brány.

3.5 Minihry

Hráči také pomohou **minihry**. K jejich spuštění je nutné mít vylepšení dané minihry alespoň na první úroveň. Poté je možné tuto minihru střetnout přímo ve hře, v podobě speciálních druhů nepřátel. Po kolizi s nimi se hra přepne do režimu minihry, ve které má hráč vždy dosáhnout určitého cíle, například co nejrychleji klikat na displej, sbírat symboly apod. Minihry jsou časově omezené, ale doba jejich trvání se dá ovlivnit šikovností hráče.

Obrázek 17: Minihry

Název minihry	Popis ovládání minihry
Balón	Hráč chytí balón, načež po něm začnou nepřátelé střílet z luků. Úkolem je tyto přilétající šípy dotykem likvidovat dříve, než propíchnou hráčův balón a hrdina tak půjde k zemi, čímž minihra končí.
Pogo tyč	Hráč chytí skákací pogo tyč a jeho úkolem je kliknout na displej v co nejmenší výšce nad zemí. Čím blíže zemi se mu podaří kliknout, tím výše ho tyč odrazí. K dispozici má 3 odrazy, poté minihra končí.
Dinosaurus	Hráč chytí dinosaura, nasedne na něj a za úkol má co nejrychleji klikat na displej. Čím rychleji kliká, tím rychleji dinosaurus utíká.
Plátěný štít	Hráč chytí štít. Držením prstu na displeji se hrdina za štít schová, puštěním prstu z displeje zase odkryje. Štít má k dispozici ukazatel výdrže, která ubývá, když je za něj hrdina schovaný a odráží tak nepřátele. Minihra končí pokud už štít nemá výdrž, nebo hrdina narazí do nepřátele, zatímco není za štít schovaný.
Raketa	Hráč chytí raketu a má za úkol ji držením prstu na displeji rozžhavit, čímž se zároveň zvyšuje její rychlost. Je zobrazen ukazatel, který ukazuje, v jaké chvíli by měl hráč prst z displeje odstranit, aby došlo k co největšímu odpalu po výbuchu rakety.
Letadlo	Hráč chytí letadlo a jeho úkolem je narazit ve vzduchu na kanystr s palivem. Po nárazu letadlo exploduje a hráč získá na rychlosti i výšce.
Plášť	Hráč chytí plášť a dostane se do horizontálního letu. Ovládá hrdinu pohybem prstu nahoru a dolu, má za úkol hrdinou posbírat co nejvíce zrychlujících symbolů a vyhnout se symbolům zpomalujícím.
Jetpack	Hráč chytí Jetpack a na displeji se začnou objevovat zrychlující i zpomalovací symboly. Úkolem je prstem pochytat co nejvíce zrychlujících symbolů a vyhnout se těm zpomalujícím.

3.6 Hrací obrazovka

Zde se odehrává to nejpodstatnější z celé hry. Hráč se vžije do role hlavního hrdiny a jeho cílem je dostat se co nejdále. Pokud možno rovnou až na svobodu. Hrdina začíná v sedě na katapultu, ze kterého vzlétne a probíjí si cestu skrze spousty nepřátel a překážek.

Obrázek 18: Hrací obrazovka

(1) Ukazatel úspěšnosti odpalu.
(2) Výběr herních výhod. Tato nabídka po odstartování kola mizí.
(3) Ukazatel rychlosti letu.
(4) Počítadlo získaných banánů.
(5) Tlačítko **Pauza** pozastaví hru.
(6) Tlačítko **Nastavení** zobrazí dialog nastavení chování hry.
(7) Ukazatel aktuálního počtu možných vzdušných úderů.

3.7 Mise

Hráče při hře doprovází různé druhy misí. Plněním misí získává herní měnu, herní výhody nebo losy do kola štěstí. Vždy jsou aktivní pouze tři mise. Jejich průběh může hráč sledovat v dialogu misí. Po úspěšném dokončení jedné z misí, se na její místo dostává mise další. Tímto způsobem se mise přidávají, dokud hráč nesplní všechny z dané kategorie. Splněním všech misí v aktuální kategorii se hráč dostává do kategorií dalších s tím, že se obtížnost jednotlivých misí vždy zvyšuje.

Obrázek 19: Mise

(1) Tlačítko **(Oukej)** zavře dialog.

(2) Tlačítko **Přeskoč** přeskočí vybranou misi za určený obnos.

(3) Zobrazení třech aktivních misí, odměny za každou z nich a jejich aktuální průběh.

(4) Ukazatel aktuální kategorie misí.

3.8 Herní úspěchy

Herní úspěchy jsou pro hráče další výzvou. Každá položka herních úspěchů je jiná a plní se jiným způsobem. Hráč musí například uletět určitou celkovou vzdálenost, nasbírat dané množství herní měny, zasáhnout několik nepřátel v jedné z miniher apod. Za plnění a získání těchto herních úspěchů hráč nedostává odměnu, slouží pouze ke zpestření hry.

Obrázek 20: Úspěchů

(1) Tlačítko **(Oukej)** zavře dialog.

(2) Tlačítko **Google Play** zobrazí herní úspěchy přímo ve službách Google Play.

(3) Zobrazení veškerých herních úspěchů a jejich aktuálního průběhu.

3.9 Kolo štěstí

V kole štěstí má hráč možnost vyhrát různé ceny, které mu pomohou s postupem ve hře. Roztočení kola stojí jeden los, který se dá získat nalezením přímo ve hře, plněním misí, nebo jej lze také v kole štěstí vyhrát zpět.

Obrázek 21: Kolo štěstí

(1) Tlačítko **Zatoč!** roztočí kolo štěstí.

(2) Ukazatel zbývajících losů.

(3) Ukazatel momentální rychlosti letu.

(4) Kolo štěstí graficky znázorňuje, co hráč vlastně vyhrál.

3.10 Herní výhody

Herní výhody jsou speciální předměty, které lze nakoupit v obchodě (záložka Goods) nebo vyhrát v kole štěstí. Předměty jsou celkem tři. Každý z nich má jinou vlastnost, a tak hráči přináší jinou výhodu. Hráč tyto předměty sbírá a před každou hrou má díky výsuvnému dialogu na pravé straně obrazovky možnost si vybrat, které použije. Hráč může použít také kombinaci těchto předmětů, nebo nepoužít žádný z nich.

Obrázek 22: Výběr výhod

Dvojité banány - V následujícím kole získá hráč dvojnásobek banánů.

Speciální šance - V následujícím kole zvyšuje šanci na střetnutí miniher.

Perfektní start - V následujícím kole bude mít hráč záruku perfektního odpalu.

3.11 Síťová hra více hráčů

Hra více hráčů se spouští tlačítkem **Multi player**, které se nachází na úvodní obrazovce. Hráč zde zadá své jméno či přezdívku a IP adresu herního serveru. Tlačítkem **Connect** se pak hráči připojí na server.

Obrázek 23: Hra více hráčů - připojení

V tomto módu hráči soupeří přímo proti sobě v tom, kdo se dostane dále. Na začátku kola, mají hráči několik vteřin na to, aby co nejlépe trefili odpal. Po skončení odpočtu oba hráči ve stejnou chvíli vystartují a hra tak začíná.

Pravidla jsou jednoduchá. Hráči v průběhu hry sbírají herní měnu jako v základním herním módu. Na konci kola se však sečtou částky obou hráčů a celý obnos putuje k hráči, který se v tomto kole dostal dále. Hráči mají k dispozici všechny speciální druhy nepřátel a také všechny minihry.

4 Závěr

Výsledkem mé práce je akční hra pro platformu Java na stolní počítač a OS Android od verze 4.0. Tvorba této hry mi zabrala spoustu času, čehož však vůbec nelituji, jelikož jsem se díky ní zlepšil v mnoha oblastech vývoje her a práce na ní mě celou dobu bavila. Prohloubila mé znalosti v jazyce Java a objevil jsem díky ní další možnosti rámce libGDX.

Největší přínos pro mě tato bakalářská práce měla hlavně v oblasti vývoje aplikací či her pro mobilní zařízení. Zacházení s více rozlišeními, optimalizace pro slabší procesory těchto zařízení atp. Také ale co se týče propojení více zařízení přes Internet, nebo například práce s animacemi včetně jejich tvorby.

Zadání práce jsem splnil. Hra obsahuje vše, co obsahovat měla (až na propojení se současně vyvíjeným portálem her, protože stále není dokončen) a ještě několik dalších vedlejších věcí navíc (jako jsou hru doprovázející mise a herní úspěchy, kolo štěstí apod.), které jsem implementoval, aby hra byla rozmanitější a nápaditější.

Hru jsem otestoval na zařízeních Nexus 7 II, Nexus 5 a HTC EVO 3D. Na prvních dvou jmenovaných zařízeních hra běží hezky, bez jakéhokoli problému. Mobilní telefon HTC EVO 3D občas problémy s vykreslením hry plynule měl, když bylo na obrazovce mnoho nepřátel a předmětů. To se však dalo předpokládat, jelikož je to už zastaralý model a tento konkrétní kus má již lecco za sebou.

V mobilních aplikacích vidím obrovský potenciál a je to směr, kterým bych se chtěl vydat. Nyní už vím, co to obnáší, vytvořit hru většího rozsahu a kolik to zabere času.

5 Reference

[1] libGDX - Getting started [online]. [cit. 2015-04-22].
Dostupné z: `http://libgdx.badlogicgames.com/documentation.html`

[2] Spine Documentation [online]. [cit. 2015-03-29].
Dostupné z: `http://esotericsoftware.com/spine-documentation`

[3] LibGDX Tutorial 10: Basic networking [online]. [cit. 2015-04-12].
Dostupné z: `http://www.gamefromscratch.com/post/2014/03/11/LibGDX-Tutorial-10-Basic-networking.aspx`

[4] Understanding JSON Schema [online]. [cit. 2015-04-13].
Dostupné z: `http://spacetelescope.github.io/understanding-json-schema/reference/object.html`

[5] Supporting Multiple Screens [online]. [cit. 2015-04-17].
Dostupné z: `http://developer.android.com/guide/practices/screens_support.html`

[6] LibGDX Tutorial 8: Audio [online]. [cit. 2015-04-17].
Dostupné z: `http://www.gamefromscratch.com/post/2013/11/19/LibGDX-Tutorial-8-Audio.aspx`

[7] Command (příkaz) [online]. [cit. 2015-03-17].
Dostupné z: `http://voho.cz/wiki/command`

[8] libgdx Tutorial: scene2d [online]. [cit. 2015-04-20].
Dostupné z: `http://steigert.blogspot.cz/2012/02/3-libgdx-tutorial-scene2d.html`

[9] Scene2d [online]. [cit. 2015-04-20].
Dostupné z: `https://github.com/libgdx/libgdx/wiki/Scene2d`

[10] Lesson: All About Sockets [online]. [cit. 2015-04-29].
Dostupné z: `https://docs.oracle.com/javase/tutorial/networking/sockets/`

6 Seznam příloh

Přiloženo je CD, které obsahuje veškeré zdrojové kódy ke kompilaci a spuštění hry na PC i zařízení s OS Android. Pro usnadnění jsou zde i předkompilované spustitelné soubory.

6.1 Struktura CD

Zdrojove_kody - adresář se zdrojovými kódy
Spustitelne_soubory - adresář se spustitelnými aplikacemi
- **Android** - obsahuje instalační soubor hry pro OS Android
- **Desktop** - obsahuje spustitelný soubor hry na PC a potřebná data
- **Server** - obsahuje spustitelný soubor serveru a potřebná data
bp_vne0005.pdf - tato bakalářská práce ve formátu pdf.